Claim(s)

1. A method of instrumenting a byte code computer program, comprising:

(a) examining the byte code;

(b) selecting portions of the byte code for instrumentation; and

(c) instrumenting the portions to provide instrumented byte code.

2. A method according to claim 1, wherein selecting the portions includes choosing portions of the byte code corresponding to at least one of: method entry, method exit, a throw, a method call, and a new line number.

3. A method according to claim 2, wherein instrumenting a portion of the byte code corresponding to a method call includes instrumenting a local line number of source code corresponding to the byte code being instrumented.

4. A method, according to claim 1, wherein instrumenting the portions includes adding calls to instrumentation runtime functions that pass parameters indicative of the portions being instrumented.

5. A method, according to claim 4, wherein at least one of the parameters that is passed includes a line number of the source code corresponding to the portion being instrumented.

6. A method, according to claim 4, wherein at least one of the parameters that is passed includes a thispointer for the method corresponding to the portion being instrumented.

7. A method, according to claim 4, wherein at least one of the parameters that is passed corresponds to at least one method parameter provided to a method containing byte code that is being instrumented.

8. A method, according to claim 7, wherein data indicative of the at least one method parameter is passed in a message buffer from an instrumentation runtime function to at least one viewer routine that displays the data to a user.

9. A method, according to claim 8, wherein the message buffer includes scalar data, array data, and object data.

10. A method, according to claim 9, further comprising:

(d) placing an object header in the message buffer; and

(e) placing an array header in the message buffer.

11. A method, according to claim 10, further comprising:

(f) limiting the message buffer to a predetermined size.

12. A method, according to claim 4, wherein data indicative of the parameters are stored in a message buffer.

13. A method, according to claim 12, wherein data from the message buffer is passed to at least one viewer routine that displays the data to a user.

14. A method, according to claim 1, further comprising:

(d) instrumenting an end of a method to provide instrumentation for handling an abort.

15. A method according to claim 1, further comprising:

(d) instrumenting a call to a native function by adding a byte code wrapper to the native function and then instrumenting the wrapper.

16. A method, according to claim 15, wherein the wrapper includes byte code corresponding to method entry and exit portions.

17. A method, according to claim 1, further comprising:

(d) instrumenting a call to a native function by providing an native assembly language thunk that captures data passed to and from the native function.

18. A method, according to claim 17, further comprising:

(e) hooking the assembly language thunk between the virtual machine and the call to the native function.

19. A method, according to claim 18, wherein hooking the assembly language thunk includes intercepting a call that provides an address for a procedure.

20. A method, according to claim 1, further comprising:

(d) providing a routine to pass data via a message stream.

21. A method, according to claim 20, further comprising:

(e) providing a data storage to store data provided via the message stream.

22. A method, according to claim 20, further comprising:

(e) providing a viewer to allow viewing at least a subset of data from the message stream as the data is being generated.

23. A method of instrumenting a computer program, comprising:

(a) examining an initial byte code representation of the program;

(b) creating a program counter mapping table corresponding to the byte code representation;

(c) selecting portions of the byte code representation for instrumentation using the program counter mapping table;

(d) instrumenting the portions by adding calls to instrumentation runtime functions at at least some of the portions; and

(e) modifying the program counter mapping table according to modifications to the byte code.

24. A method according to claim 23, wherein selecting the portions includes choosing portions of the initial intermediate representation corresponding to at least one of: method entry, method exit, a throw, a method call, and a new line number.

25. A method, according to claim 23, wherein instrumenting the portions includes adding calls to instrumentation runtime functions that pass parameters indicative of the portions being instrumented.

26. A method according to claim 23, further comprising:

(f) instrumenting a call to a native function by adding a byte code wrapper to the native function and then instrumenting the wrapper.

27. A method, according to claim 26, wherein the wrapper includes byte code instrumentation corresponding to method entry and exit portions.

28. A method, according to claim 23, further comprising:

(f) instrumenting a call to a native function by providing an native assembly language thunk that captures data passed to and from the native function.

29. A method, according to claim 28, further comprising:

(g) hooking the assembly language thunk between the virtual machine and the call to the native function.

30. A method, according to claim 29, wherein hooking the assembly language thunk includes intercepting a call that provides an address for a procedure.

31. A method, according to claim 23, further comprising:

(g) following examining an initial byte code representation of the program, registering each of the methods and corresponding line numbers thereof.

32. A method, according to claim 31, wherein registering each of the methods and corresponding line numbers thereof facilitates determining the source code being executed during run time debugging.

33. A method of uniquely identifying an object in an object oriented programming language, comprising:

(a) obtaining a unique identifier corresponding to the object;

(b) creating a data structure having a least a first and a second storage location;

(c) storing an identifier for the object in the first storage location; and

(d) storing the unique identifier in the second storage location.

34. A method, according to claim 33, wherein obtaining the unique identifier includes generating a hash code.